Gensh R, Rafiev A, Romanovsky A, Garcia A, Xia F, Yakovlev A.

Architecting Holistic Fault Tolerance.

***In: 18th IEEE International Symposium on High-Assurance Systems Engineering (HASE 2017)*. 2017, Singapore: IEEE Computer Society.**

DOI link to article:

https://doi.org/10.1109/HASE.2017.13

Date deposited:

27/06/2017

2017 IEEE 18th International Symposium on High Assurance Systems Engineering

Architecting Holistic Fault Tolerance

Rem Gensh, Ashur Rafiev, Alexander Romanovsky
CSR
Newcastle University
Newcastle upon Tyne, UK

Alessandro Garcia
Informatics Department
PUC-Rio
Rio de Janeiro, Brazil

Fei Xia, Alex Yakovlev
School of EEE
Newcastle University
Newcastle upon Tyne, UK

***Abstract*—The efficiency and maintainability of fault tolerance mechanisms in a computer system has typically not been a major topic of concern, mostly because fault tolerance is a non-functional system requirement. This paper proposes a Holistic Fault Tolerance architecture, based on a centralised fault tolerance management, with related functionality distributed across the entire system. The most suitable error detection and error recovery strategies for a given application are chosen by a special crosscutting controller depending on error rates, system performance and resource utilisation requirements. We discuss the motivation for introducing this holistic fault tolerance architecture and reason about its benefits from the point of view of optimal system operation and improved maintainability. The advantages and possible implementation challenges of the proposed approach are demonstrated by a real-world application.**

Keywords—architecture; fault tolerance; croscutting concerns; performance; operation modes;

I. INTRODUCTION

Faults and failures are unavoidable in computer systems. To prevent catastrophic consequences of these failures, computer systems must be both reliable and safe, ensuring overall system dependability. In addition, they should be optimal in terms of resource utilisation because performance and energy efficiency are important factors for the systems regardless of scale from embedded devices to data centres. It is also crucial to provide an easy way to maintain and modify the system components to decrease outage time, improve developer's understanding of the system and reduce associated costs. The same demands are applied to fault tolerance (FT) mechanisms of the system.

During system design a lot of effort is made to provide high cohesion and loose coupling of the system components [1]. This approach is appropriate for functional properties and for business logic, because the same functionality may be placed in one unit and intricate details may be hidden from other units. However, it makes system-wide FT less understandable and optimisable, since FT mechanisms are hidden as well. Such an approach causes components to be designed maximally safe with the costs not always contributing to system-wide FT, precluding the possibility of centralised monitoring and dynamic tuning of the system based on the interplay between performance, resource utilisation and reliability.

This work is supported by the EPSRC PRiME EP/K034448/1 and MRC Newton001 MR/M026388/1 projects.

To deal with these issues we propose the holistic fault tolerance (HFT) architecture, which allows developers to control the system FT in a global crosscutting manner. Our approach is called *holistic*, because we propose to consider an entire application during the design of the system's FT. The FT property is chosen as the main part of the concept, since it is an important crosscutting concern of the system, which can affect other non-functional properties of the system. The vision of Holistic Fault Tolerance was proposed in our recent paper [2] where we presented a novel approach to system FT taking into consideration non-functional characteristics of the system, such as reliability, performance and energy efficiency. HFT assumes that the FT mechanisms across the entire system are managed by a central component, allowing the developer to reason about certain error detection and error recovery strategies at the system scale, and not at the scale of separate components. The HFT architecture does not imply the alteration of established FT techniques [3]. In contrast, it demonstrates how these techniques can be applied for the design and implementation of more efficient computer systems by reasoning about the system FT holistically, rather than concentrating on individual components only. While the HFT approach is considered general and not restricted to any application domain, our main focus is given to component-based software architectures, since they are more suitable for the scope of this article. Therefore, this type of software architecture will be assumed in the rest of the paper.

The main contribution of this paper is an HFT architecture, which allows the designer to have centralised access to the FT functionality of the system and to tune non-functional properties such as reliability, performance and resource utilisation. In addition, we consider a general method to facilitate the design of the HFT architecture. The goal of this study is to demonstrate that HFT is able to monitor and dynamically adjust the entire system to achieve optimal operation without uncontrolled reliability deterioration.

According to academic [4] and industrial [5] studies, FT is a crosscutting concern for computer systems. In this way, during the design and implementation of FT functionality the main focus should be made system-wide, rather than on components. It is more convenient to centralise FT-related code in order to facilitate the modularity of the system, since the relevant FT functionality will be coordinated by a single module, unit or component, simplifying the understanding and access to the FT mechanisms. The main dilemma is how to

1530-2059/17 $31.00 © 2017 IEEE
DOI 10.1109/HASE.2017.13

create a crosscutting component-controller implementing system-wide FT scenarios and at the same time to avoid over-complicating the system architecture by binding this centralised controller to all crucial components of the system.

In previous work [2] we presented two reasons for introducing HFT. The first is efficient system-wide operation. Without a centralised coordination a system may consist of a set of locally optimal components, which do not provide global efficiency. The other relates to system maintenance – system-wide FT is not easily understandable and modifiable without a holistic approach. These remain the fundamental motivations for the current work where we present further development of the HFT architecture that has been made so far, consider all its elements in details and describe the techniques that could be applied for the implementation of the HFT architecture. The rest of the paper is organised as follows. Section II provides background support for the chosen study. Section III describes all elements of the HFT architecture. A practical way to apply the HFT and its benefits are demonstrated by the case study in Section IV. Concluding remarks are given in Section V.

II. Backgound and existing work

This section provides an analysis of the existing approaches to system structuring and FT management, and considers the following issues: a centralised management of FT, system architectures based on goal-seeking behaviour, modular FT architectures and operation modes.

Centralisation of FT management is considered in [6], which provides the notion of guardian – a special global exception handler for a distributed system. Such systems require the communication and coordination of exception handlers, since each participating process should invoke the correct handler. In the guardian model, the correct handler for a process is chosen by the guardian according to the application defined recovery rules allowing the guardian to orchestrate the recovery action of each involved process. However, the implementation of reliable broadcast with participating processes involves scalability and performance overheads.

The system architecture can be considered from the goal-achieving point of view. Brooks describes the architecture of layered control systems developed for mobile robots [7]. Levels of competence and layers of control are applied to solve each small decomposed subproblem. Each next level of competence offers more complex behaviour. For each level of competence there is a corresponding layer of control. A higher layer augments lower layers of the control system, but the lower layers still produce the results. The idea of the Teleo-Reactive (TR) programs is presented in [8]. To apply this approach, the developer should specify the goal and define the actions to be performed in case of changes in a constantly monitored environment. Monitoring is implemented as continuous computation of the parameters and conditions for the actions. However, it requires a lot of computations.

Aspect Oriented Programming (AOP) is a promising paradigm intended to improve the modularity of systems by separation of crosscutting concerns. It is achieved by extending the program code behaviour in certain points, without modifying the code itself. In [9] the quantitative assessment of exception handling as aspects is provided. The author considers the benefits of using AOP for modularisation of exception handling by lexical separation of exception handling code from normal application code given that the changes in AOP code will be less intrusive and simpler. However, there is no possibility to represent global properties of exception control flows. Research [10] investigates whether AOP facilitates the modularization of exception handling mechanisms. The main result of the study is that AOP will not improve FT in the system with bad architecture. However, it is able to facilitate the structure of well-designed systems by separating normal and exceptional activities of the system. Study [11] presents feasibility and evaluation of using AOP to avoid tangling of software implemented hardware FT (SIHFT) code and main functionality code. According to the experimental results AOP is convenient for the programs with SIHFT.

Operation mode (OM) is a functional state of the system. It defines which system functionality is available at this point in time. In other words, OM determines the link between the system state and available functionality, since the capabilities that are available in one mode may not be available in another mode. In [12] authors promote the notion of a mode as a partition of the system state space and as a convenient method for modular specification of large state machines. There are serial and parallel modes. Serial mode means that the system could only be in one mode in one instant of time, whereas the parallel relationship assumes that the system is in all of the available parallel modes simultaneously. Modal systems [13] are defined as an abstract specification of the modes and mode transition. It is claimed that OMs are very common in real-time systems, for example a deadline could be dependable on OM.

III. Holistic Fault Tolerance Architecture

In the previous sections we considered the issues relating to efficient system operations and the convenience of FT maintainability. To address these problems, we propose the HFT architecture blueprint. This architecture assumes that the application is build out of components whose responsibility is to deliver the main system functionality. The core of this architecture is a special component called the HFT controller, which is supported by a number of HFT agents. These together ensure dependable and optimal system operation. In addition, they provide a clear view of the system FT mechanisms. The HFT controller coordinates system-wide FT with the assistance of the HFT agents that simplify the implementation and improve the scalability of the HFT controller. Each HFT agent acts as an intermediary between the HFT controller and one or more system components.

The *HFT controller* is a crosscutting unit, which coordinates FT strategies and analyses the performance of the entire system. These tasks are mainly performed with the assistance of the HFT agents, which obtain all required information from the monitored system components and pass it to the HFT controller. Moreover, the HFT controller initiates fault handling and reconfiguration of the entire system after detecting certain erroneous conditions and checking error rates. In this case, apart from the HFT agents' help, the HFT controller utilises public interfaces of critical system components in order to adjust the reliability, quality of service,

performance and energy consumption of the system. However, it should not be aware of the inner structure or encapsulated information of the monitored components because in this case it will be very complex for maintenance and understandability. This is the reason why the knowledge of the HFT controller about the system should be restricted by the general structure of the system and the performance characteristics and average resource utilisation requirements of the system components. The HFT controller should know about the ties between the HFT agents and system components. In all cases when the implementation details of the monitored system components are required to perform some action, the HFT controller should use the HFT agents, which are responsible for their areas of the system and able to provide all required information. Depending on this data and stored HFT policies, the HFT controller will make system adjustments and reconfigurations.

An *HFT agent* is a special object monitoring one or more system components. The introduction of HFT agents is aimed at improving the scalability of the HTF architecture. The HFT agents are aware of the implementation details of these components and have the possibility and right of intervention to the control flow inside the principal functions of monitored components in order to check results, perform error detection and error recovery, evaluate performance and supress exception raising. The HFT agent is not aware of the entire system structure, because its goal is to monitor only parts of the system and pass the up-to-date information to the HFT controller. If the HFT agent monitors more than one component, errors could be detected based on concurrent analysis of two components. In this case, error recovery could affect both components as well.

The HFT controller works with all available HFT agents. In case of error in an observed component, the HFT agent could request the HFT controller for a suitable error recovery. In addition, the HFT agent should detect the states in the monitored component that may cause an error, and propagate this information to the HFT controller. To reduce HFT controller complexity, we propose to use discrete enumerations by the HFT controller whenever it is possible. For instance, quality of the operation or result of the function could be presented by the following enumeration: Error, Low Quality, Medium Quality and High Quality. The task to map from the component-specific data to the simplified data suitable for the HFT controller is performed by the HFT agents.

For the case of direct interaction between the HFT controller and the system components, the latter should provide special public interfaces, which are used to adjust component settings, to choose operation mode and/or to perform fault handling by reconfiguring the component. These interfaces enable the HFT controller to tune the reliability and performance of the entire system. All actions that are not expressed via public interfaces and require the amendment of the component behaviour should be done via the HFT agents. Thus, the HFT controller does not have strong ties to the system components, but it still has global knowledge about the entire system.

FT mechanisms in the given architecture are distributed across the entire system, but coordinated centrally by the HFT controller. In some cases, it is beneficial to introduce redundancy in FT mechanisms in such a way that the same error could be handled by the component itself and by the HFT agent. The decision on suitable error handling scenarios will be made by the HFT controller depending on the current system state. Such an approach provides the flexibility in the choice of the optimal error recovery scenario. Some errors will be handled by both the component itself and the HFT agent. Only part of the system components needs to be involved in the HFT mechanisms. It makes sense to use only critical components that globally affect the system operation or could be reconfigured in terms of performance or resource usage. It is definitely more convenient to implement these system components to be "HFT-ready" providing all necessary interfaces and preparing them to work with the HFT controller and HFT agent. On the other hand, when the components do not provide such interfaces, for example legacy components, the developers can implement special wrappers or adapters.

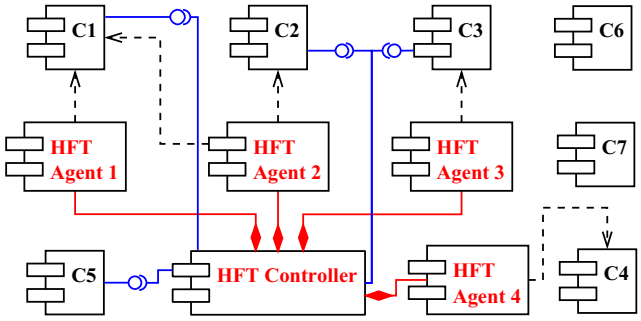

Fig. 1. The HFT architecture

A system designer should choose which components of the system will participate or will be included in the HFT behaviour and which components will just provide their functionality without being affected by the HFT controller and agents. The system components are classified into four groups. The first group (C1, C2 and C3 in Fig. 1) includes the components that are monitored by one or more HFT agent and provide the interface for the HFT controller. The second group (C4 in Fig. 1) includes components that are monitored by the HFT agent/s only and do not provide interfaces for the HFT controller. Components that only provide the interface for the HFT controller are in the third group (C5 in Fig. 1). This means that for the given components it does not make sense to observe their inner operation, but they provide good flexibility in tuning performance and resource utilisation. The last group (C6 and C7 in Fig. 1) includes components that just provide their functionality and are not part of the HFT scheme.

The HFT architecture includes operation modes, considering the interplay between reliability, performance and energy consumption. OMs can be applied for the entire system and for separate components. The HFT controller is the most suitable place to control and choose the optimal OM for the system components. Let us consider the following example. Two components are performing some operation. To finish one cycle, the chunk of data should be processed by one component and put in the queue. The second component checks the queue. When there is a chunk of data, it takes this chunk for

processing. To balance the loading of the components we can specify how to distribute computer resources between these two components by OM assignation and how to balance the data chunk queue. This can be elegantly done in the HFT controller. HFT agents monitor these components and supply the information to the HFT controller, so that the HFT controller is able to increase or decrease the quality of service for each component on-the-fly. OM can also be considered as a graceful degradation for the system when some component fails or requires restart. This idea provides the possibility of fault handling with the assistance of the HFT controller, which performs system reconfiguration by choosing suitable modes for the system components.

The next section presents a case study demonstrating practical usage of the HFT architecture for real applications.

IV. CASE STUDY

The case study is centred on a software application for the recognition of UK car number plates. The main goal of the application is to demonstrate the practical use of the HFT approach and evaluate the proposed HFT architecture. It also helps to explain the stages of the design and implementation of the HFT architecture of the system. The application is not intended to compete with industrial solutions, but shows how the HFT architecture can be employed during real-world software development processes. The implementation is made in Java with AspectJ [14] AOP extension.

After uploading the image file to the application, the first step of number plate recognition is initial image processing, which includes resolution and contrast adjustment, localisation of the number plate cutout and elimination of the rotation. There are two algorithms for this. The first is light and fast, whereas the second is complex, but more reliable. Depending on the system state, image size and image quality the most suitable algorithm will be chosen for the current image. The next step is optical character recognition with three algorithms available, which differ in quality and performance. It is chosen dynamically which algorithm to use for the current number plate cutout. In some cases, two or three algorithms may be launched concurrently to provide reliable character recognition. We applied third-party algorithms for both steps.

The FT and performance of the application are managed by the HFT controller with the assistance of two HFT agents, which have access to encapsulated information of the application components. The first agent measures the performance of crucial components. The second agent is responsible for error handling. We applied AOP for the implementation of the HFT agents. *Around* advice [15] is used to implement custom behaviour before and after the invocation of the target method. If necessary, the method result can be substituted. It should be noted that the erroneous state could be defined not only by catching the exception, but also by checking special conditions or values of certain variables in monitored components. AOP can significantly simplify the development of the HFT agents. The developer does not need to change the observed components, which almost eliminates the possibility of introducing bugs in existing code.

At present two operation modes are available, namely reliability and performance. The HFT controller analyses monitoring information from the HFT agents and sets the most suitable operation mode through the public interface provided by the components. The mode is chosen depending on the current performance and reliability requirements. In certain cases, the HFT controller can assign performance mode for one component and reliability mode for another component or vice versa if this action will improve the efficiency of the application.

V. CONSCLUSION

In this paper we presented a Holistic Fault Tolerance architecture, which implies that system-wide FT strategies and resource distribution in the system are coordinated by the HFT controller with the assistance of a number of HFT agents. This architecture and its associated design methods have been applied to a case study application, which has passed initial experimental validation and analyses. Comparative studies with other approaches will follow in the immediate future.

REFERENCES

[1] W. P. Stevens, G. J. Myers and L. L. Constantine, "Structured design," in *IBM Systems Journal*, vol. 13, no. 2, pp. 115-139, 1974.

[2] R. Gensh, A. Romanovsky, A. Yakovlev. 2016. “On structuring holistic fault tolerance,” in *Proceedings of the 15th International Conference on Modularity*. ACM, New York, USA, 130-133.

[3] T. Anderson, P. A. Lee, Fault tolerance, principles and practice, Prentice/Hall International. 1981.

[4] R. Alexandersson, P. Öhman and M. Ivarson, “Aspect Oriented Software Implemented Node Level Fault Tolerance,” in *Proceedings of the 9th IASTED International Conference on Software Engineering and Applications*, Phoenix, Arizona, USA, 2005.

[5] Microsoft Patterns & Practices Team. (2009). *.NET Application Architecture Guide, 2nd Edition*. Microsoft Press.

[6] R. Miller, A. Tripathi. 2004. “The Guardian Model and Primitives for Exception Handling in Distributed Systems”. *IEEE Trans. Softw. Eng*. 30, 12 (December 2004), 1008-1022.

[7] R. Brooks, “A robust layered control system for a mobile robot,” in *IEEE Journal on Robotics and Automation*, 1986.

[8] N. J. Nilsson, “Teleo-reactive programs for agent control,” *Journal of Artificial Intelligence Research*, vol. 1, no. 1, pp. 139-158, 1993.

[9] N. Cacho, "Supporting Maintainable Exception Handling with Explicit Exception Channels," PhD thesis, Lancaster University, 2009.

[10] F. C. Filho, A. Garcia and C. M. F. Rubira, "Extracting Error Handling to Aspects: A Cookbook," *2007 IEEE International Conference on Software Maintenance*, Paris, 2007, pp. 134-143.

[11] S. Karol, N. A. Rink, B. Gyapjas, J. Castrillon. 2016. “Fault tolerance with aspects: a feasibility study,” in *Proceedings of the 15th International Conference on Modularity*. ACM, New York, USA, 66-69.

[12] F. Jahanian and A. K. Mok. 1994. Modechart: A Specification Language for Real-Time Systems. *IEEE Trans. Softw. Eng*. 20, 12, 933-947.

[13] F. L. Dotti, A. Iliasov, L. Ribeiro, A. Romanovsky. 2009. “Modal systems: Specification, refinement and realisation,” in *Proceedings of the 11th International Conference on Formal Engineering Methods.* Lecture Notes in Computer Science, vol. 5885, Springer, 601–619.

[14] R. Laddad. *AspectJ in Action*. Manning, 2003.

[15] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. Loingtier, J. Irwin, "Aspect-oriented programming," In *Proceedings of the 11th ECOOP*, LNCS 1271, pages 220–242, 1997.